TMS320DM3730 Software Developers Guide

Translate this page to Translate Show original

Contents

Welcome to the TMS320DM3730 Software Developer's Guide

Thanks you for choosing the TMS320DM3730 Evaluation Module (EVM) for your application. The purpose of this guide is to get you going with developing software for the TMS320DM3730 on a Linux development host only.

Note! This Software Developer's Guide (SDG) supports version 4.xx of the TMS320DM3730 DVSDK which is only for Linux host development.

Note! This guide assumes you have already followed the Quick Start Guide (QSG) for setting up your EVM and installing the Digital Video Software Development Kit (DVSDK). If you have not done this yet, please do so now before continuing. You can find a hard copy contained with your EVM. Alternatively you can find the QSG PDF and various other documentation in the 'docs' directory of the DVSDK installation directory.

Note! All instructions in this guide are for Ubuntu 10.04 LTS. At this time, it is the only supported Linux host distribution for development.

Note! In previous DVSDK releases there has been a *Getting Started Guide* explaining how to set up the DVSDK. This document replaces and extends the Getting Started Guide for DVSDK 4.xx.

Throughout this document there will be commands spelled out to execute. Some are to be executed on the Linux development host, some on the Linux target and some on the u-boot (bootloader) prompt. They are distinguished by different command prompts as follows:

```
host $ <this command is to be executed on the host>
target # <this command is to be executed on the target>
u-boot :> <this command is to be executed on the u-boot prompt>
```

Starting your software development

Your DVSDK should be installed before you continue. Throughout this document it will be assumed you have an environment variable *DVSDK* which points to where your DVSDK is installed. You can set it as follows (the following assumes that DVSDK was installed at default location):

```
host $ export DVSDK="${HOME}/ti-dvsdk_dm3730-evm_xx_xx_xx_xx"
```

Setting up the DVSDK

The DVSDK comes with a script for setting up your Ubuntu 10.04 LTS development host as well as your target boot environment. It is an interactive script, but if you accept the defaults by pressing return you will use the recommended settings. This is recommended for first time users. Note that this script requires internet access as it will update your Ubuntu Linux development host with the packages required to develop using the DVSDK. Before executing the script make also sure that the SD card received with the EVM or an SD card prepared as described in the section below "How to create an SD card" is inserted in the EVM SD card reader.

Execute the script using:

```
host $ ${DVSDK}/setup.sh
```

If you accepted the defaults during the setup process, you will now have set up your development host and target to:

1. Boot the Linux kernel from your development host using TFTP. On your development host the Linux kernel is fetched from */tftpboot* by default.
2. Boot the Linux file system from your development host using a Network File System (NFS). On your development host the Linux target file system is located at *${HOME}/targetfs*
3. Minicom is set up to communicate with the target over RS-232. If you want to use a windows host for connecting to the target instead, see the #Setting_up_Tera_Term section.

If you start minicom on your Linux development host using *minicom -w* (or Tera Term on Windows) and power cycle the EVM, Linux will boot.

After Linux boots up, login into the target using **root** as the login name.

Note! The Matrix Application Launcher GUI is not launched automatically in the development filesystem. If you would like to start it, execute the following command on the target board:

```
target # /etc/init.d/matrix-gui-e start
```

If your kit includes an LCD display, the first time the Matrix GUI is executed from NFS, you'll go through a LCD touchscreen calibration process. The calibration process is important as other application in additional to the Matrix GUI require calibration to run successfully. You can also run the calibration manually without starting the Matrix GUI by executing the following command on the target board:

```
target # ts_calibrate
```

If the Matrix is running, make sure you have terminated the Matrix GUI before running any other applications from the command line:

```
target # /etc/init.d/matrix-gui-e stop
```

Note! if you select the "Primary display output" to DVI, then you might be required to have mouse support to work with graphical user interface, you must first clear these variables prior to running the demos to enable the mouse:

target # export QWS_MOUSE_PROTO=

target # export TSLIB_TSDEVICE=

Writing your own "Hello World!" application and executing it on the target

This section shows how to create/build an application on your host development PC and execute a basic Linux application on your booted target filesystem.

1. Create your own work directory on the host PC and enter it:

```
host $ mkdir ${HOME}/workdir
host $ cd ${HOME}/workdir
```

2. Create a new C source file:

```
host $ gedit helloworld.c
```

Enter the following source code:

```
#include <stdio.h>

int main()
{
    printf("Hello World!\n");
}
```

Save the file and exit.

3. Create a basic makefile:

```
host $ gedit Makefile
```

Enter the following:

```
# Import the variables from the DVSDK so that you can find the DVSDK components
include ${DVSDK}/Rules.make

helloworld:
# Make sure that you use a tab below
	$(CSTOOL_PREFIX)gcc -o helloworld helloworld.c
```

Save the file and exit. Note that the gap before $(CSTOOL_PREFIX)gcc corresponds to a tab. If it is filled with spaces instead you will get build errors.

4. Make sure the $DVSDK variable is still set using:

```
host $ echo $DVSDK
```

This command should print your DVSDK installation directory. If it doesn't, you will have to set it again as described in the beginning of this document. Compile the application:

```
host $ make helloworld
```

As a result, an executable called `helloworld` is generated in `${HOME}/workdir`

5. You now have your own application, but you need to create a directory and copy it to your NFS exported filesystem to make it visible by the target:

```
host $ mkdir ${HOME}/targetfs/home/root/dm3730
host $ cp helloworld ${HOME}/targetfs/home/root/dm3730
```

6. On your target this application will be accessible from `/home/root/dm3730/helloworld`. Execute it on your target:

```
target # /home/root/dm3730/helloworld
```

You should now see the following output:

```
Hello World!
```

Congratulations! You now have your own basic application running on the target.

Running the pre-installed applications on the target file system

The filesystem comes with a number of prebuilt applications (which can be rebuilt inside the DVSDK). This section shows how to execute those applications in the provided filesystem.

- A system wide loadmodule script is provided in the filesystem. This script loads the various TI kernel modules as part of the Linux init process which are needed by the various applications provided in the filesystem. The file can be found in the following location on the target:

```
target # vi /etc/init.d/loadmodule-rc
```

The script leverages a new feature of CMEM 2.0 to use general purpose heaps instead of specific pools. More information on this can found in the CMEM Overview on TI's embedded processor wiki page.

You can use the load/unload/restart parameter to load and unload the various TI kernel modules at any time by executing the following on the target:

```
target # /etc/init.d/loadmodule-rc start|stop|restart
```

Running the DVSDK demos from the command line

The DVSDK multimedia demos can be launched from the Matrix application launcher. There is a copy installed in your NFS mounted target filesystem as well. First enter the demo directory on your target:

```
target # cd /usr/share/ti/dvsdk-demos/
```

Then, load the kernel modules:

```
target # /etc/init.d/loadmodule-rc restart
```

You can quickly view information on options available to the demos using the -h option. For example:

```
target # ./encode -h
target # ./decode -h
```

Alternatively you can refer to even more detail in the documentation in these files:

```
target # cat encode.txt
target # cat decode.txt
```

After reading up on the options, you can execute the demos from the command line like this:

```
target # ./encode <options>
target # ./decode <options>
```

Note that there are multimedia clips installed under `/usr/share/ti/data` on the target.

If you prefer to run the demos using a graphical user interface (instead of the textual one), you can pass in the -o option to do so.

Running the DSPLink examples

The DSPLink comes with a few sample application. To run them enter this directory on the target

```
target # cd /usr/share/ti/ti-dsplink-examples/
```

Execute the following script to run the example application

```
target # ./ti-dsplink-examples-run.sh
```

The target terminal window will output the results of the examples executed. The examples can be run individually (vi ti-dsplink-examples-run.sh for proper parameters to individual examples).

Running the C6Accel apps

The C6Accel package comes with a small test application benchmarks all the DSP kernel APIs for fixed point and floating point calculations. To run the application, enter the following directory on the target:

```
target # cd /usr/share/ti/c6accel-apps/
```

Load the C6Accel specific kernel modules:

```
target # ./loadmodules_omap3530_c6accel.sh
```

Execute the following command to run the example application

```
target # ./c6accel_app
```

The application benchmarks all the DSP kernel API calls in C6Accel and writes the benchmark data to file (benchmarking.txt) in the /usr/share/ti/c6accel-apps directory. To view the file, execute

```
target # vi /usr/share/ti/c6accel-apps/benchmarking.txt
```

Running the DMAI apps

The *Davinci Multimedia Application Interface* (DMAI) comes with small sample applications (including source code). To run them enter this directory on the target:

```
target # cd /usr/share/ti/ti-dmai-apps/
```

Then, load the kernel modules:

```
target # /etc/init.d/loadmodule-rc restart
```

The DVSDK comes with DMAI, and the following example invocations are known to work. For more information on how to run DMAI applications, refer to the DMAI user guide shipped with the DMAI installation in the DVSDK. Hit Ctrl-C to terminate any example if needed.

Decode examples

To decode a H.264 BP encoded video to a YUV file execute:

```
target # ./video_decode_io2_dm3730.x470MV  -c h264dec \
-i /usr/share/ti/data/videos/davincieffect_480p30.264 -o h264_test_output.yuv
```

To decode a MPEG4 SP encoded video to a YUV file execute:

```
target # ./video_decode_io2_dm3730.x470MV  -c mpeg4dec \
-i /usr/share/ti/data/videos/davincieffect_480p30.m4v -o mpeg4_test_output.yuv
```

To decode a MPEG2 MP encoded video to a YUV file execute:

```
target # ./video_decode_io2_dm3730.x470MV  -c mpeg2dec \
-i /usr/share/ti/data/videos/davincieffect_480p30.m2v -o mpeg2_test_output.yuv
```

To decode a JPEG encoded image to a YUV file execute:

```
target # ./image_decode_io1_dm3730.x470MV -c jpegdec \
-i /usr/share/ti/data/images/remi003_422i.jpg -o jpeg_test_output.yuv
```

To decode a AAC encoded audio

```
target # ./audio_decode_io1_dm3730.x470MV -c aachedec \
-i /usr/share/ti/data/sounds/davincieffect_lc.aac -o aac_test_output.pcm
```

Encode examples

To encode 30 frames of resolution 720x480 from a YUV file to an H.264 BP encoded file execute:

```
target # ./video_encode_io1_dm3730.x470MV -c h264enc \
-i h264_test_output.yuv -o output.264 -r 720x480 -n 30
```

To encode 30 frames of resolution 720x480 from a YUV file to an MPEG4 SP encoded file execute:

```
target # ./video_encode_io1_dm3730.x470MV -c mpeg4enc \
-i mpeg4_test_output.yuv -o output.m4v -r 720x480 -n 30
```

To encode a YUV file to a JPEG encoded file execute:

```
target # ./image_encode_io1_dm3730.x470MV -c jpegenc -e encode \
-i jpeg_test_output.yuv -o Output.jpeg -r 720x576 \
--iColorSpace 3 --oColorSpace 1
```

The input parameters depends on the configuration of the input YUV file. In this case, the input file color space format is YUV422 ILE

To know more about the color space values as supported by the application, execute:

```
target # ./image_encode_io1_dm3730.x470MV -h
```

Loopback examples

To do video loopback from composite input to LCD output execute:

```
target # ./video_loopback_copy_dm3730.x470MV -O lcd -I composite
```

To do video loopback from composite input to DVI output execute:

```
target # ./video_loopback_copy_dm3730.x470MV -O dvi -I composite
```

To do video loopback from composite input to LCD output without copying the video frame execute:

```
target # ./video_loopback_dm3730.x470MV -O lcd -I composite
```

To resize D1 from composite input to CIF on LCD output execute:

```
target # ./video_loopback_resize_dm3730.x470MV -r 352x288 -O lcd -I composite
```

Display examples

To display a test pattern at 720P 60fps on DVI output execute:

```
target # ./video_display_dm3730.x470MV -y 5 -O dvi
```

Running the Digital Video Test Bench (DVTB)

The DVTB allows you to tweak codec parameters using an interactive text based user interface and observe the result. The DVTB is particularly useful because it is allows to modify not only codec base parameters but also codec extended parameters.

To run it enter this directory on the target

```
target # cd /usr/share/ti/dvtb/
```

Then, load the kernel modules:

```
target # /etc/init.d/loadmodule-rc restart
```

First launch the dvtb

```
target # ./dvtb-r
```

At the dvtb prompt, enter the following commands to decode the first 30 frames in the davincieffect_480p30.m2v file and write them to file output.yuv. This example illustrates how to modify codec base parameters.

```
<DVTB> $ setp engine name encodedecode
<DVTB> $ setp viddec2 maxWidth 720
```

```
<DVTB> $ setp viddec2 maxHeight 480
<DVTB> $ setp viddec2 numFrames 30
<DVTB> $ getp viddec2
<DVTB> $ func viddec2 -s /usr/share/ti/data/videos/davincieffect_480p30.m2v -t output.yuv
```

After completion of the decode process type "enter" to return to the "<DVTB> $" prompt. Then, enter the following command to exit DVTB application

```
<DVTB> $ exit
```

There are scripts provided for some commonly used DVTB tests. These are under scripts folder under /usr/share/ti/dvtb. The scripts which have a codec specific name use extended parameters specific to that codec. For example scripts/h264enc1.dvs uses extended parameters for the h264 encoder codec.

To run the scripts execute

```
target # ./dvtb-r -s scripts/<script-name>.dvs
```

Please modify the scripts as needed with the correct path to the multimedia clip used.

Running the Qt/Embedded examples

The Qt embedded comes with some examples applications. To see the examples that are available, check out this directory on the target:

```
target # cd /usr/bin/qtopia/examples
target # ls
```

Execute the following command to run Qt/e calendar example application.

```
target #  cd /usr/bin/qtopia/examples/richtext/calendar
target # ./calendar -qws -geometry 320x200+50+20
```

After you see the calendar interface, hit **CTRL-C** to terminate it or click on the **X** on the top right hand corner of the calendar window.

Running GStreamer pipelines

Load the kernel modules before running the below gstreamer pipelines:

```
target # /etc/init.d/loadmodule-rc restart
```

The DVSDK comes with GStreamer, and the following pipelines are known to work. You can construct your own pipelines, see Gstreamer pipeline. Refrain from using Ctrl-C to terminate gst pipelines as un-expected results may occur. Pipelines will terminate on their own.

This pipeline decodes AAC audio :

```
target # gst-launch filesrc location=/usr/share/ti/data/sounds/davincieffect_lc.aac \
! typefind ! TIAuddec1 ! alsasink -v
```

This pipeline decodes 720P MPEG-4 video:

```
target # gst-launch filesrc location=/usr/share/ti/data/videos/davincieffect.mpeg4 \
! typefind ! mpeg4videoparse ! TIViddec2 padAllocOutbufs=TRUE ! queue ! tidisplaysink2  -v
```

This pipeline decodes 720P H.264 video:

```
target # gst-launch filesrc location=/usr/share/ti/data/videos/davincieffect.264 \
! typefind ! h264parse ! TIViddec2 padAllocOutbufs=TRUE ! queue ! tidisplaysink2 -v
```

This pipeline decodes 720P MPEG-2 video:

```
target # gst-launch filesrc location=/usr/share/ti/data/videos/davincieffect.m2v \
! typefind ! mpegvideoparse ! TIViddec2 padAllocOutbufs=TRUE ! queue ! tidisplaysink2 -v
```

This pipeline decodes 720P MP4 video :

```
target # gst-launch filesrc location=/usr/share/ti/data/videos/davincieffect_h264_aac.mp4 \
! qtdemux name=demux demux.audio_00 ! queue max-size-buffers=8000 max-size-time=0 \
max-size-bytes=0 ! TIAuddec1 ! alsasink demux.video_00 ! queue \
! TIViddec2 padAllocOutbufs=TRUE ! queue ! tidisplaysink2 -v
```

This pipeline encodes 720P video color pattern generated by videotestsrc in MPEG-4:

```
target # gst-launch videotestsrc num-buffers=100 ! \
'video/x-raw-yuv,width=1280,height=720,format=(fourcc)UYVY' ! TIVidenc1 codecName=mpeg4enc \
engineName=codecServer ! filesink location=sample.m4v
```

This pipeline encodes 720P video color pattern generated by videotestsrc in H.264:

```
target # gst-launch videotestsrc num-buffers=100  ! \
'video/x-raw-yuv,width=1280,height=720,format=(fourcc)UYVY' ! TIVidenc1 codecName=h264enc \
engineName=codecServer ! filesink location=sample.264
```

NOTE: tidisplaysink2 is an experimental sink and intended to replace TIDmaiVideoSink. For more information run gst-inspect tidisplaysink2 on target to see various properties.

Running 3D Graphics Demos

The 3d graphics demos can be executed from the Matrix application launcher. There is a copy installed in your NFS mounted target filesystem as well. First enter the graphics demo directory on your target:

```
target # cd /usr/bin/SGX/demos/Raw
```

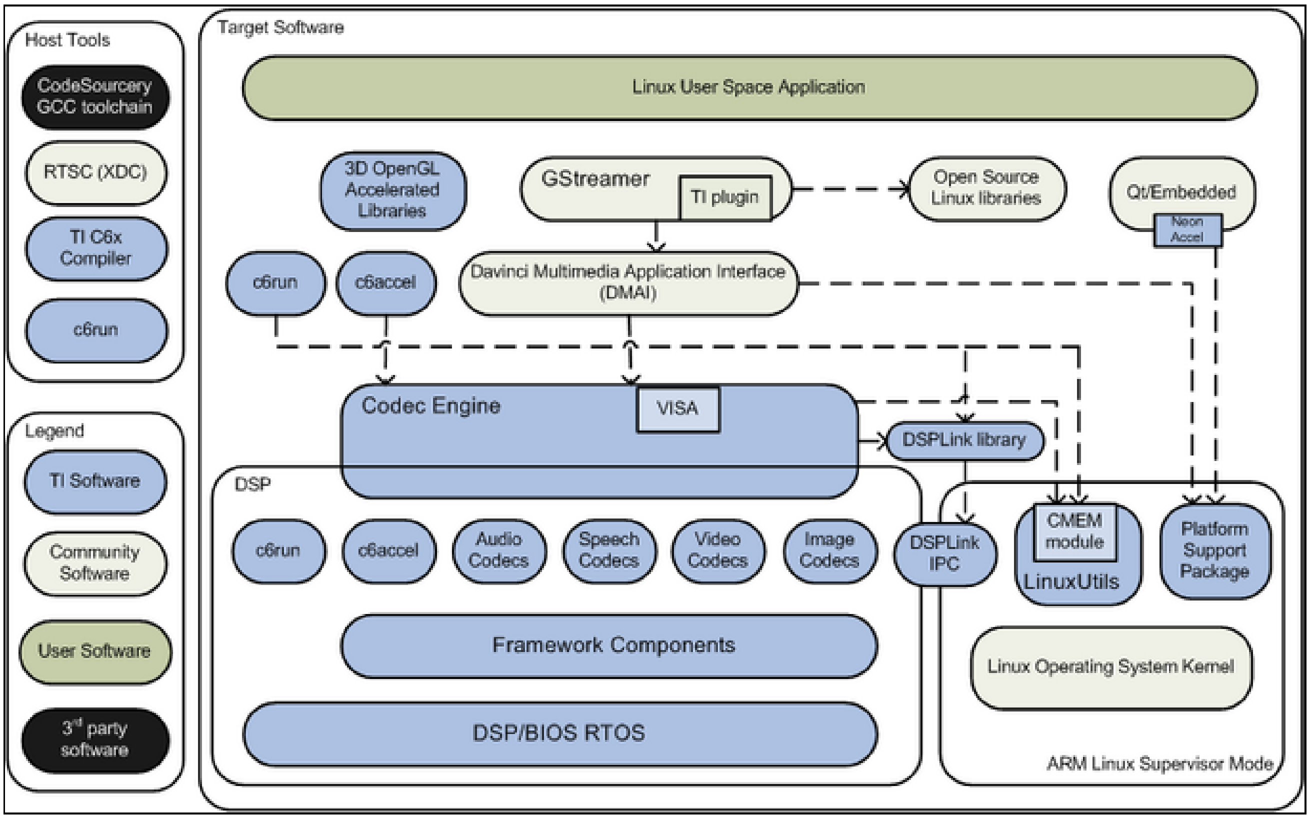
This folder contains both the OPENGL ES1.0 and ES2.0 demos. To execute, perform the following on the target

```
target # ./OGLESFur
```

This will execute the OGLESFur demo. Press 'q' on the host machine console window (TeraTerm or HyperTerminal or Minicom) to stop the demo.

Similarly, other demos could be executed. Press 'q' to stop the demo

DVSDK software overview and support



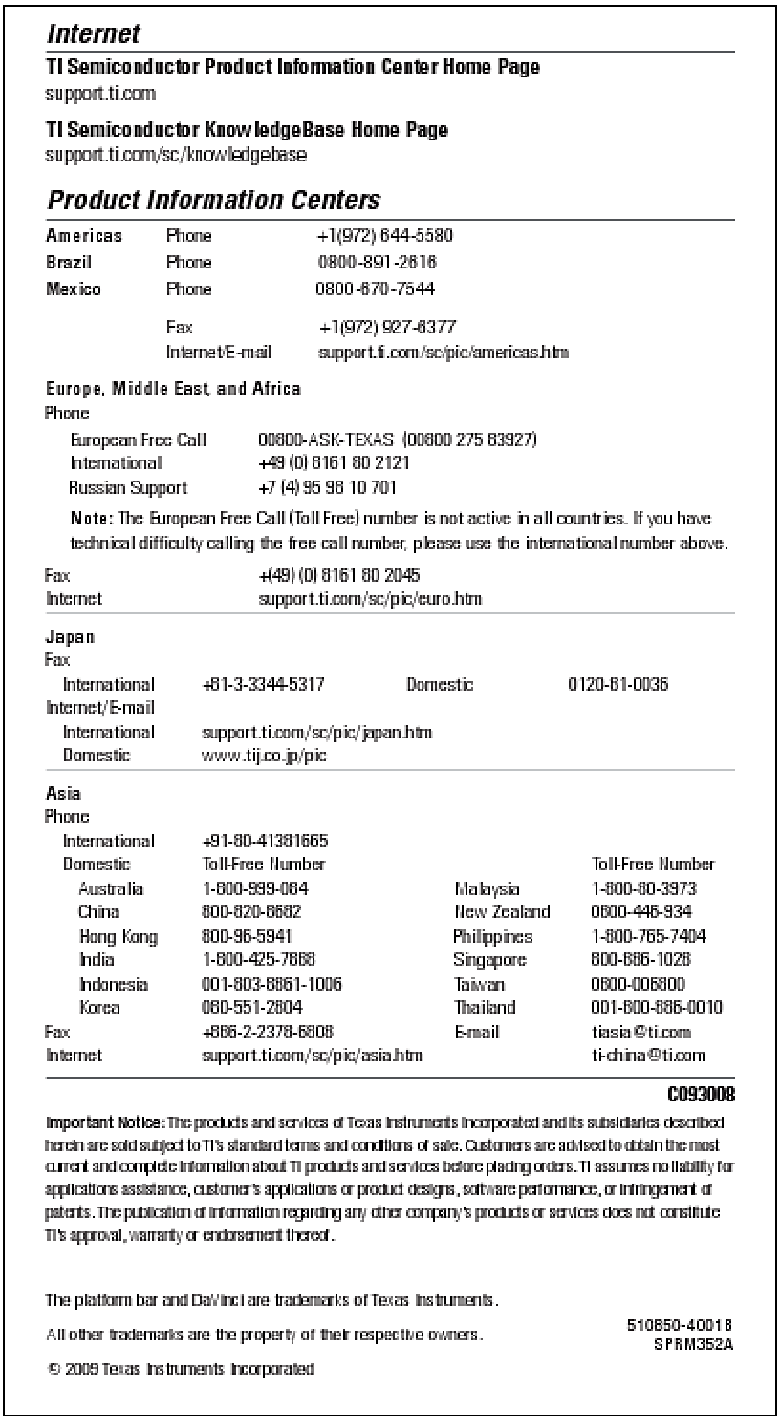
Overview of the DVSDK Software stack

The DVSDK contains many software components. Some are developed by Texas Instruments (*blue* in the diagram above) and some are developed in and by the open source community (*grey* in the diagram above). TI contributes, and sometimes even maintains, some of these open source community projects, but the support model is different from a project developed solely by TI. The table below lists how to get support for each component.

Component(s)	Type	Support
DVSDK demos, Codec Engine, Multimedia Codecs, Platform Support Package, Neon accelerated Qt, Accelerated OpenGL, Framework Components, DSP/BIOS, DSPLink, c6accel etc.	Texas Instruments Developed Software	In addition to the support channels listed at #TI Worldwide Technical Support you can use the following support channels: The community Linux forum is monitored by TI support and engineering teams and can be used for asking questions about development using this SDK. The multimedia codecs forum is a separate forum for discussing the

		multimedia codecs. If you have a bug tracking number (starting with *SDOCM*) you can track the issue using the SDOWP bug tracking web access
Davinci Multimedia Application Interface	Open Source Project	DMAI community project
GStreamer	Open Source Project	GStreamer community project
GStreamer plugin for accelerated multimedia	Open Source Project	gst-ti community project
Qt/Embedded	Open Source Project	Qt/Embedded community project
RTSC (XDC)	Open Source Project	RTSC community project
Linux kernel	Open Source Project	Linux kernel community project

TI Worldwide Technical Support



Internet

TI Semiconductor Product Information Center Home Page
support.ti.com

TI Semiconductor KnowledgeBase Home Page
support.ti.com/sc/knowledgebase

Product Information Centers

Americas	Phone	+1(972) 644-5580
Brazil	Phone	0800-891-2616
Mexico	Phone	0800-670-7544
	Fax	+1(972) 927-6377
	Internet/E-mail	support.ti.com/sc/pic/americas.htm

Europe, Middle East, and Africa

Phone

European Free Call	00800-ASK-TEXAS (00800 275 83927)
International	+49 (0) 8161 80 2121
Russian Support	+7 (4) 95 98 10 701

Note: The European Free Call (Toll Free) number is not active in all countries. If you have technical difficulty calling the free call number, please use the international number above.

Fax	+(49) (0) 8161 80 2045
Internet	support.ti.com/sc/pic/euro.htm

Japan

Fax

International	+81-3-3344-5317	Domestic	0120-81-0036

Internet/E-mail

International	support.ti.com/sc/pic/japan.htm
Domestic	www.tij.co.jp/pic

Asia

Phone

International	+91-80-41381665		
Domestic	Toll-Free Number		Toll-Free Number
Australia	1-800-999-084	Malaysia	1-800-80-3973
China	800-820-8682	New Zealand	0800-446-934
Hong Kong	800-96-5941	Philippines	1-800-765-7404
India	1-800-425-7888	Singapore	800-886-1028
Indonesia	001-803-8861-1006	Taiwan	0800-006800
Korea	080-551-2804	Thailand	001-800-886-0010
Fax	+886-2-2378-6808	E-mail	tiasia@ti.com
Internet	support.ti.com/sc/pic/asia.htm		ti-china@ti.com

C093008

Important Notice: The products and services of Texas Instruments Incorporated and its subsidiaries described herein are sold subject to TI's standard terms and conditions of sale. Customers are advised to obtain the most current and complete information about TI products and services before placing orders. TI assumes no liability for applications assistance, customer's applications or product designs, software performance, or infringement of patents. The publication of information regarding any other company's products or services does not constitute TI's approval, warranty or endorsement thereof.

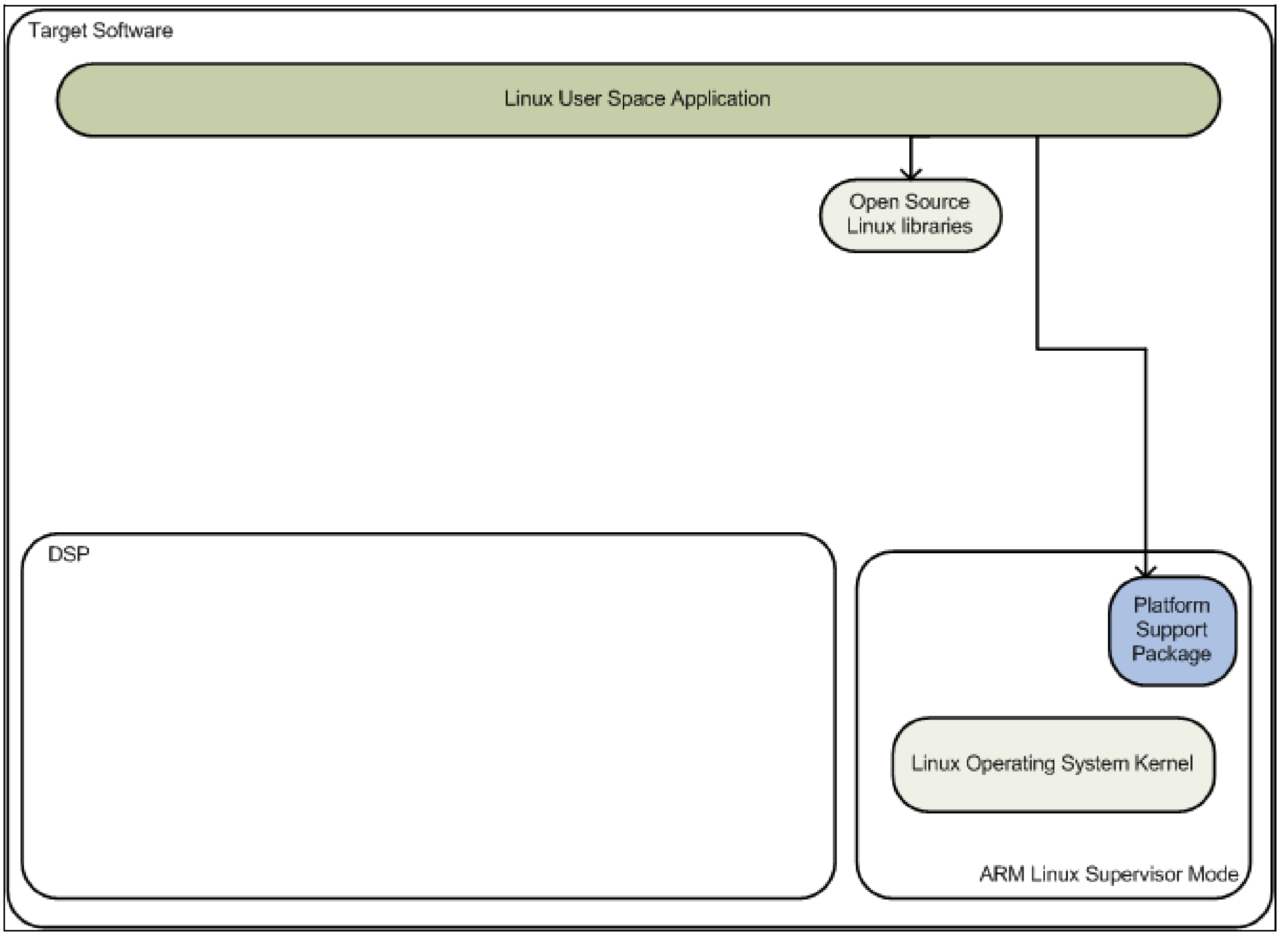
The platform bar and DaVinci are trademarks of Texas Instruments.

All other trademarks are the property of their respective owners.

© 2009 Texas Instruments Incorporated

510850-40018
SPRM352A

Creating a Linux application


Target Software
Linux User Space Application
Open Source Linux libraries
DSP
Platform Support Package
Linux Operating System Kernel
ARM Linux Supervisor Mode

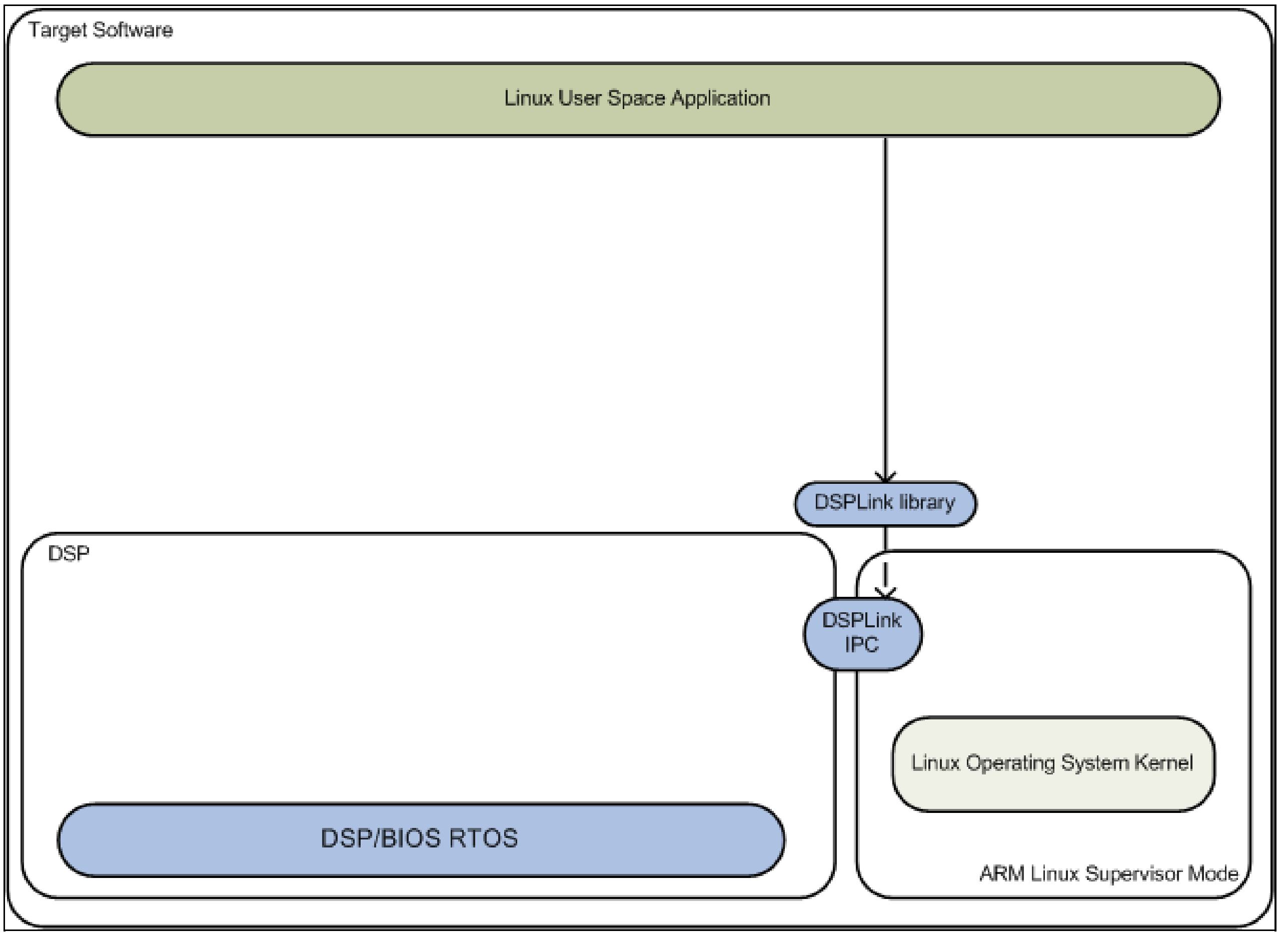
Overview of a basic Linux application component usage

While creating a basic Linux application you are typically using the following components of the stack (the rest are greyed out above):

Component	Purpose in this application	Location in the DVSDK
GCC toolchain	Cross compiler for generating ARM Linux binaries.	linux-devkit directory under the DVSDK
Open Source Linux libraries	Provides libraries such as libpng, libusb, libz, libcurl etc.	linux-devkit/arm-arago-linux-gnueabi/lib and linux-devkit/arm-arago-linux-gnueabi/usr/lib/
Platform Support Package	Provides device drivers for the EVM and documentation and examples to support them.	psp
Linux kernel	The Linux kernel with the PSP device drivers	psp/linux-kernel-source

You can find examples all over the web on how to write this type of application. The PSP examples are a good reference on how to access the peripheral drivers specific to this platform.

Creating a DSPLink application


Target Software
Linux User Space Application
DSPLink library
DSP
DSPLink IPC
Linux Operating System Kernel
DSP/BIOS RTOS
ARM Linux Supervisor Mode

Overview of a DSPlink application component usage

DSPLink is foundation software for the inter-processor communication across the GPP-DSP boundary. It provides a generic API that abstracts the characteristics of the physical link connecting GPP and DSP from the applications. It eliminates the need for customers to develop such link from scratch and allows them to focus more on application development. This software can be used across platforms:

- Using SoC (System on Chip) with GPP and one DSP.
- With discrete GPP and DSP.

DSPLink provides several features and capabilities that make it easier and more convenient for developers using a multi-core system:

- Provides a generic API interface to applications
- Hides platform/hardware specific details from applications
- Hides GPP operating system specific details from applications, otherwise needed for talking to the hardware (e.g. interrupt services)
- Applications written on DSPLink for one platform can directly work on other platforms/OS combinations requiring no or minor changes in application code
- Makes applications portable
- Allows flexibility to applications of choosing and using the most appropriate high/low level protocol
- Provides scalability to the applications in choosing only required modules from DSPLink.

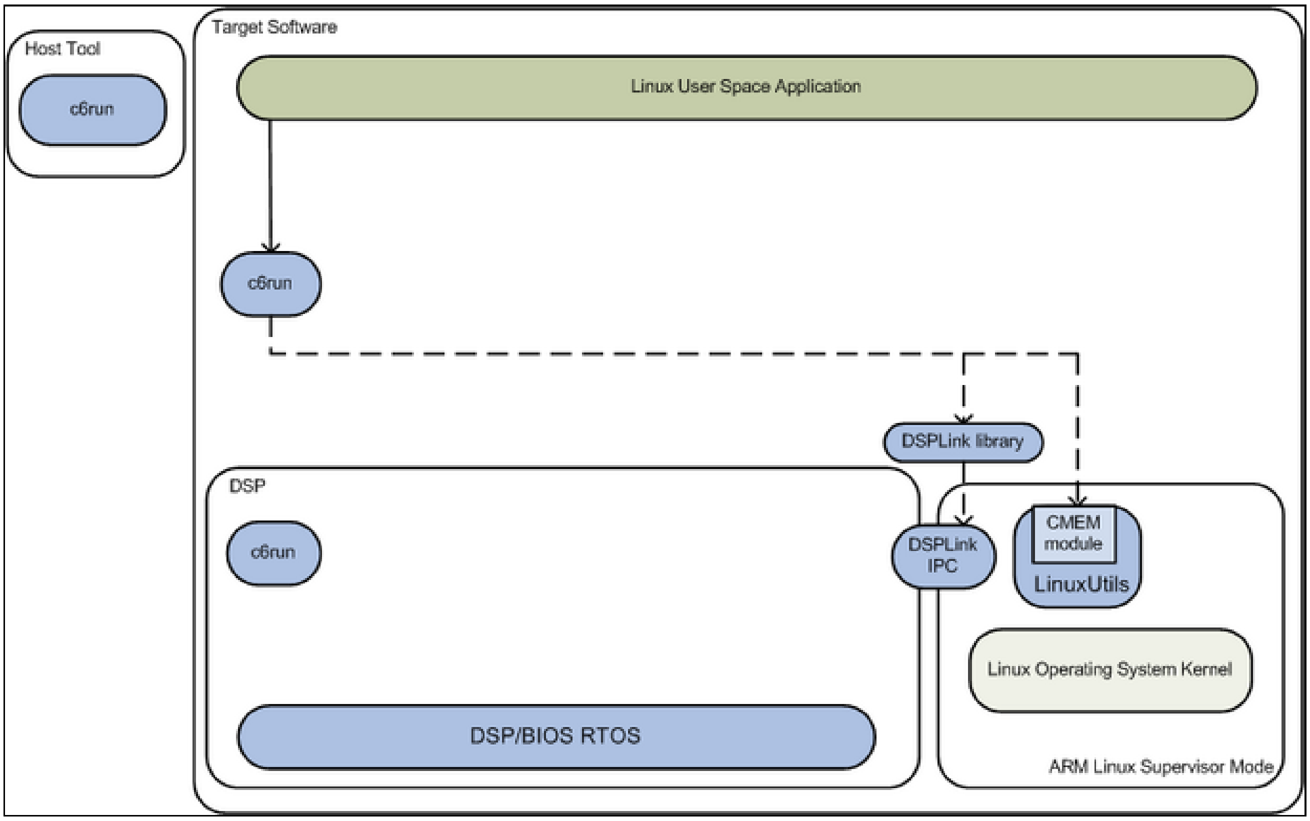
In addition to the components used for the basic Linux app, these are used (and the rest is greyed out in the diagram above):

Component	Purpose in this application	Location in the DVSDK
DSP/BIOS	Real-Time Operation System for TI DSPs	dspbios_x_xx_xx_xx
DSPLink	GPP to DSP processor communication link for passing messages and data in multiprocessor systems	dsplink_x_xx_xx_xx
C6000 Code Generation Tools	TI DSP code generation tools	cgt6x_x_x_xx

Good application examples to start from include:

- The sample applications (dsplink_x_xx_xx_xx/dsplink/gpp/src/samples and dsplink_x_xx_xx_xx/dsplink/dsp/src/samples) provide simpler and smaller examples on how to use DSPLlink.
- On some platforms, an Audio SOC example application (audio_soc_example_x_xx_xx_xx) is delivered that leverages DSPLink to perform audio processing and output the data via the DSP peripheral drivers. This example illustrates how DSP-side peripheral drivers run in conjunction with the Linux kernel application on a ARM processor. More information on this example can be found at http://processors.wiki.ti.com/index.php/Audio_Soc_example

Creating a C6Run application



Overview of a basic C6Run application component usage

The C6Run package is to ease initial development and loading of DSP code for ARM developers who are familiar with building applications for the Linux OS using an ARM GCC cross-compiler. The project consists of two main components:

1. A build system to create back-end libraries from the various TI software technologies and the code of the C6Run project itself
2. Front-end scripts that wrap the TI C6000 code generation tools in a GCC-like interface and also make use of the back-end build system to create ARM-side executables or libraries that transparently make use of the DSP.

There are two uses of the C6Run project, exposed through two different front-end scripts. They are called **C6RunLib** and **C6RunApp**.

- **C6RunLib** works to build a static ARM library from C source files that can be linked with an ARM application and provide access to the DSP when library functions are called. This allows the user to keep portions of the application on the ARM and move other portions to the DSP.

- **C6RunApp** tool acts as a cross-compiler for the DSP, allowing portable C applications to be rebuilt for the C6000 DSP core of various Texas Instruments heterogeneous (ARM+DSP) processors. The C6RunApp front-end consists of a single script, called c6runapp-cc. This use of this script matches, as much as possible, the use of GCC. It can compile C code to C6000 object files and link the C6000 object files into an application. When performing linking operations, the tool makes use of a number of steps (including linking using the C6000 code generation tools) to create an ARM-side executable from the DSP object files.

In addition to the components used for the Linux app, these are used (and the rest is greyed out in the diagram above):

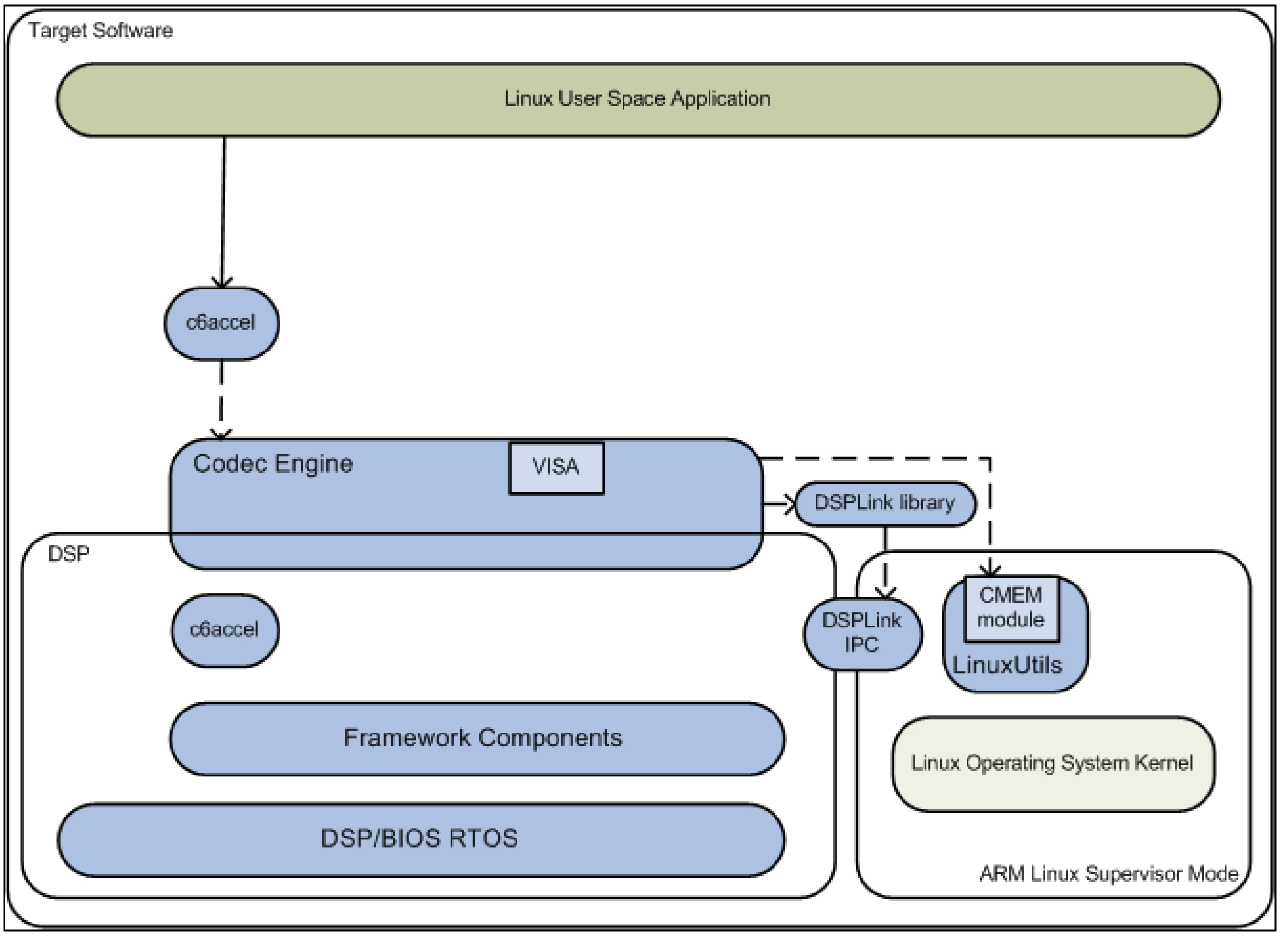
Component	Purpose in this application	Location in the DVSDK
LinuxUtils	Linux specific utilities for Framework Components, used for allocating physically contiguous memory (CMEM module, see this wiki topic for more information) for sharing data between the ARM and DSP.	linuxutils_xx_xx_xx_xx
RTSC (XDC)	Tool required to configure and build DSP/BIOS real-time kernel for the DSP.	xdctools_xx_xx_xx_xx
Local Power Manager	TI power management package (not required for all platforms)	local_power_manager_x_xx_xx_xx
DSP/BIOS	Real-Time Operation System for TI DSPs	dspbios_x_xx_xx_xx
DSPLink	GPP to DSP processor communication link for passing messages and data in multiprocessor systems	dsplink_x_xx_xx_xx
C6000 Code Generation Tools	TI DSP code generation tools	cgt6x_x_x_xx

Good application examples to start from:

- The C6Run package contains sample applications to test/validate the functionality. The applications are located in the c6run_xx_xx_xx_xx/examples and the c6run_xx_xx_xx_xx/test directories. Each example includes full source and standard makefiles.
- There is a QT-based fractal example that leverages C6Run to perform the fractal computation on the DSP. Information on how to build and run the example can be found at: C6Run QT Fractal Example

For more information on C6Run visit the TI Embedded Processors wiki, C6Run Project Page.

Creating a C6Accel application



Overview of a basic C6Accel application component usage

The C6Accel package wraps key DSP software kernels in an xDAIS algorithm which can be invoked from the ARM side using simple API calls. C6Accel can be used in a plug and play like any other codec used for encoding and decoding audio and video streams. C6Accel is built in the codec engine compliant IUniversal framework and can be used on various DSP only and ARM + DSP devices.

The purpose of C6Accel is to provide the ARM user with the compute power of the DSP on computational intense tasks like running Color Space Conversion, Filtering or Image/Signal Processing algorithm. The library of DSP kernels wrapped in C6Accel are optimized for performance on the DSP core and would allow the ARM user to use the DSP as an accelerator for their application. By using these routines, the ARM developer can develop a more compelling application by achieve execution speeds considerably faster than equivalent C code written on ARM. In addition, by providing ready-to-use DSP kernels, C6Accel can significantly shorten the ARM application development time.

The benefits of using C6Accel include:

1. **Ready to use kernels**: Library of Optimized DSP kernels wrapped in a single package. Reduces learning curve and time to market.
2. **Easy to interface**: ARM side API library abstracts complexities while invoking DSP functionality from ARM application
3. **Easy Portability**: Fully compatible with most TI C6x devices
4. **Efficient multiple call execution**: Capabilty to chain kernel calls using single call to codec engine
5. **Easy Evaluation of DSP performance:** DSP kernel Benchmarks (cycle and code size) provided in C6Accel aid in evaluating performance that can be leveraged from the DSP and make informed decisions while developing applications
6. **Parallel processing:** Asynchronous calling mode enables parallel processing on DSP and ARM
7. **Simple Template to add functionality on DSP:** SoC developers can explore maximum flexibility by using C6Accel algorithm as a template to add custom compute intense functionality on the DSP that can be accessed from the ARM.

In addition to the components used for the Linux app, these are used (and the rest is greyed out in the diagram above):

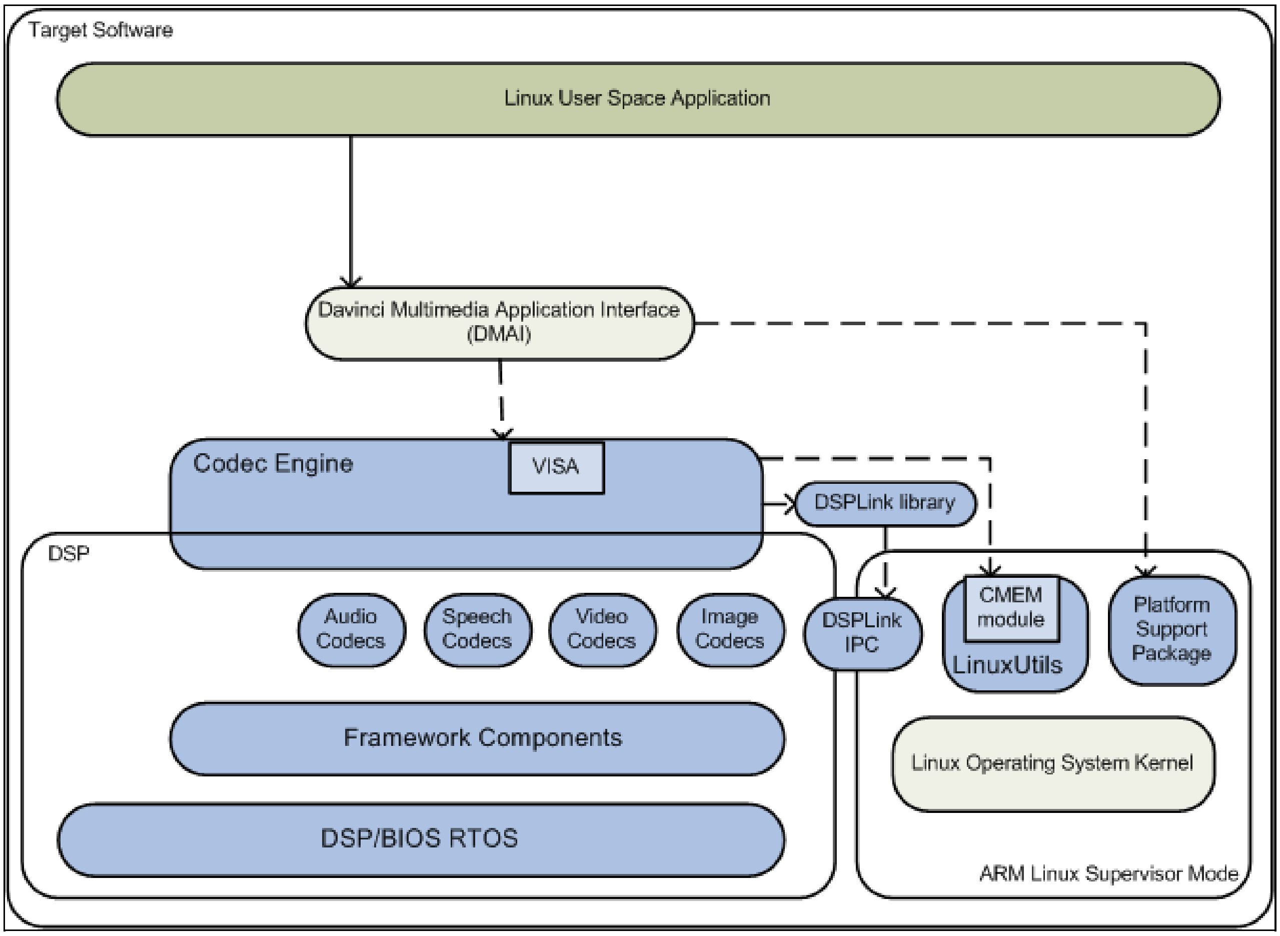
Component	Purpose in this application	Location in the DVSDK
Codec Engine	Cross platform framework for the applications invoking multimedia codecs and other algorithms.	codec_engine_xx_xx_xx_xx
LinuxUtils	Linux specific utilities for Framework Components assisting with resource allocation of DMA channels (EDMA module), physically contiguous memory (CMEM module, see this wiki topic for more information) and allows the codecs to receive completion interrupts of various coprocessor resources (IRQ module).	linuxutils_xx_xx_xx_xx
RTSC (XDC)	Tool used to configure Codec Engine, Framework Components and multimedia codecs for your application.	xdctools_xx_xx_xx_xx
XDAIS	TI Algorithm Interface Standard used for algorithm standardization which is used by various other components including Codec Engine	xdais_x_xx_xx_xx
DSPLINK	GPP to DSP processor communication link for passing messages and data in multiprocessor systems	dsplink_x_xx_xx_xx

Good application examples to start from include:

- The C6Accel contains a sample application to test/validate the functionality. The application is located in the c6accel_xx_xx_xx_xx/soc/app directory.

For more information on C6Accel visit C6Accel: ARM access to DSP software

Creating a DMAI multimedia application



Overview of a DMAI application component usage

The Davinci Multimedia Application Interface (DMAI) is a thin utility layer on top of Codec Engine and the Linux kernel. The benefits of using DMAI include:

1. DMAI and it's sample applications are written to adhere to the XDM 1.x semantics for multimedia codecs. Codec Engine facilitates the invocation of the codecs, but DMAI provides the semantics to make the codecs plug and play.
2. DMAI wraps the Linux device drivers in a multimedia function focused API, shielding you from the rapid progress of the Linux kernel, increasing your portability.
3. The DVSDK demos and gst-ti plugin are written on top of DMAI. If you can make a codec work with the DMAI sample applications, it will most likely work in these applications too.

In addition to the components used for the Linux app, these are used (and the rest is greyed out in the diagram above):

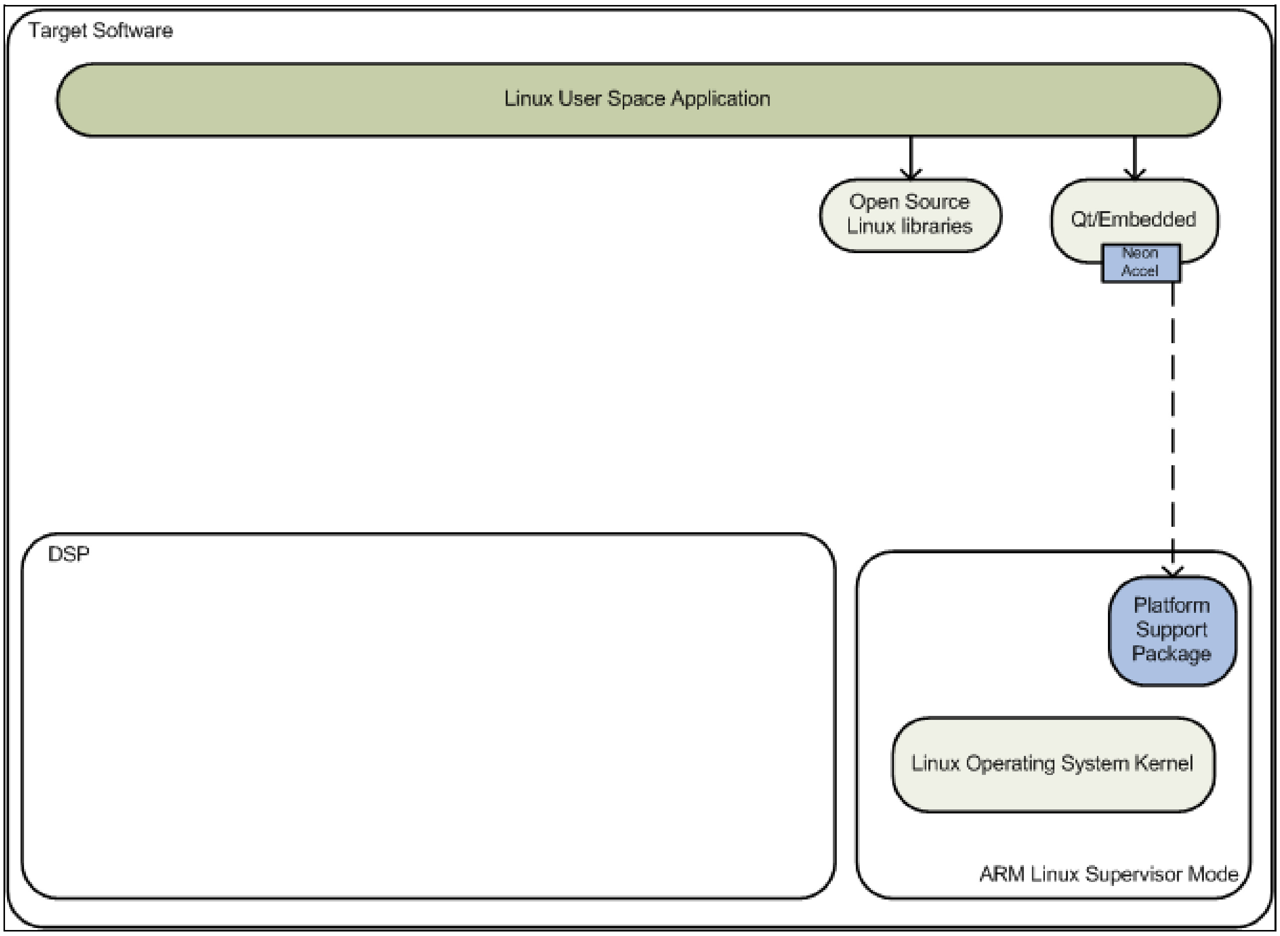
Component	Purpose in this application	Location in the DVSDK
Codec Engine	Cross platform framework for the applications invoking multimedia codecs and other algorithms.	codec_engine_xx_xx_xx_xx

Framework Components	Cross platform framework for servicing resources to algorithms.	framework_components_xx_xx_xx_xx
LinuxUtils	Linux specific utilities for Framework Components assisting with resource allocation of DMA channels (EDMA module), physically contiguous memory (CMEM module, see this wiki topic for more information) and allows the codecs to receive completion interrupts of various coprocessor resources (IRQ module).	linuxutils_xx_xx_xx_xx
Davinci Multimedia Application Interface	Multimedia application utility layer	dmai_xx_xx_xx_xx
Multimedia Codecs	Compression and decompression of multimedia data	codecs_<platform>_xx_xx_xx_xx
RTSC (XDC)	Tool used to configure Codec Engine, Framework Components and multimedia codecs for your application.	xdctools_xx_xx_xx_xx

Good application examples to start from include:

- The DMAI sample applications (dmai_xx_xx_xx_xx/packages/ti/sdo/dmai/apps) provide simpler and smaller examples on how to use DMAI to create a multimedia application.
- If applicable for your device, the DVSDK demos located in the dvsdk_demos_xx_xx_xx_xx of the $(DVSDK) installation directory. These use DMAI to provide a full multimedia application. However, the application does not support A/V sync; if this feature is required GStreamer is a better option.

Creating a Qt/Embedded application


Target Software
Linux User Space Application
Open Source Linux libraries
Qt/Embedded
Neon Accel
DSP
Platform Support Package
Linux Operating System Kernel
ARM Linux Supervisor Mode

Overview of a Qt/Embedded application component usage

Qt/Embedded is a Graphical User Interface toolkit for rendering graphics to the Linux framebuffer device, and is included in this kit. The base Qt toolkit on the other hand renders the graphics to the X11 graphical user interface instead of to the basic framebuffer.

In addition to the components used for the basic Linux app, these are used (and the rest is greyed out in the diagram above):

Component	Purpose in this application	Location in the DVSDK
Qt/Embedded	Provides a Graphical User Interface toolkit	linux-devkit/arm-arago-linux-gnueabi/usr/lib/libQt*

See the Qt Reference Documentation on various API's and its usages. You can also download some Qt/e example applications from Qt Examples web page.

Compiling an application

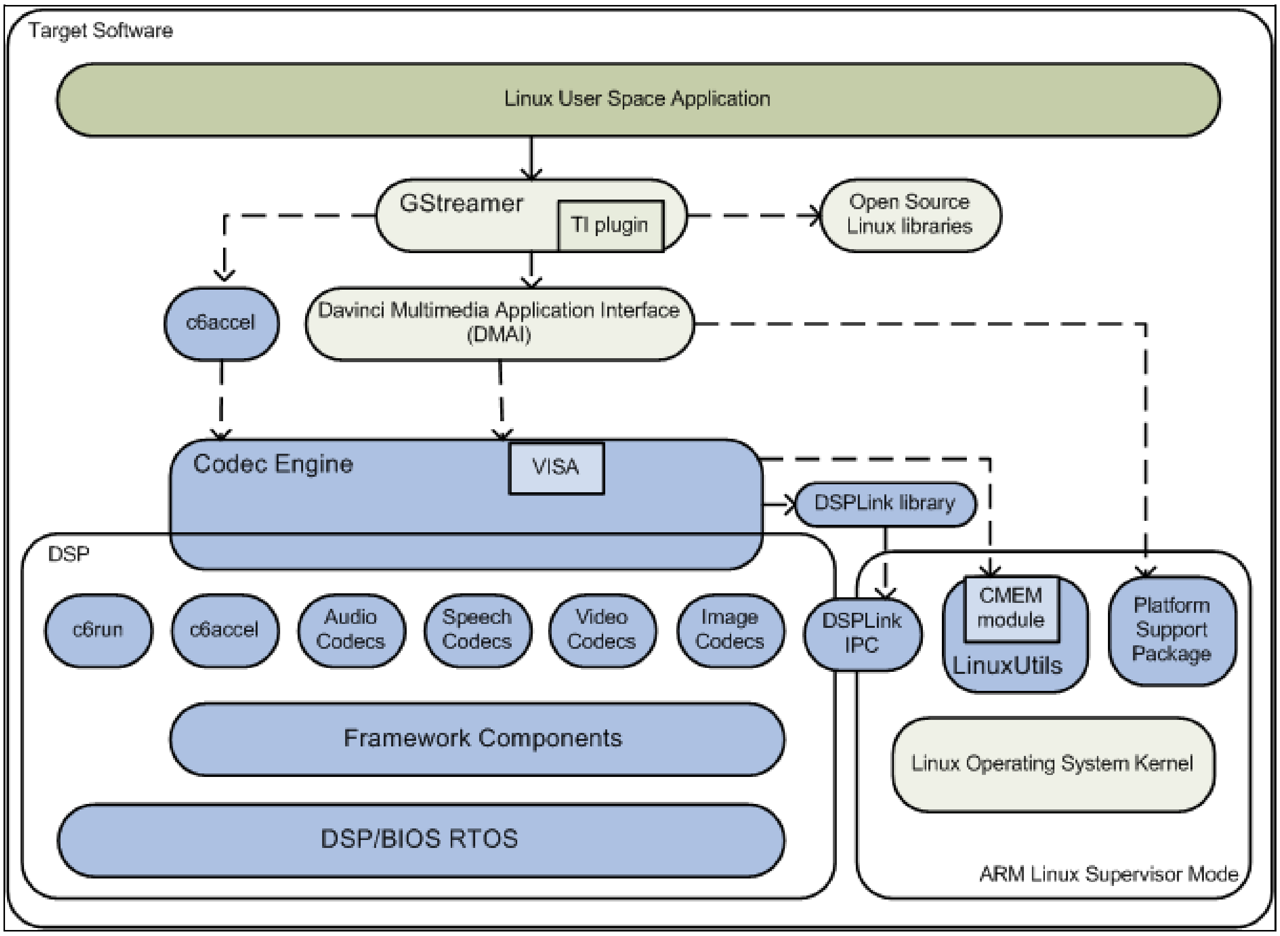
DVSDK Linux development kit includes the Qt/Emebedded host tools and development header and libraries.

1. First, configure your cross compilation environment #Setting_up_cross_compilation_environment.

2. Next, follow the typical Qt/e recommended method for cross compiling your application on host.

```
host $ cd <directory where your application is>
host $ qmake -project
host $ qmake
host $ make
```

Creating a GStreamer application



Overview of a GStreamer application component usage

GStreamer is an open source multimedia framework which allows you to construct pipelines of connected plugins to process multimedia content. There is a plugin which accelerates multimedia using DMAI and Codec Engine.

Compared to creating an application directly on top of DMAI you get the advantage of A/V sync and access to many useful open source plugins which e.g. allows you to demux avi-files or mp4-files. The downside is increased complexity and overhead.

In addition to the components used for the DMAI app, these are used (and the rest is greyed out in the diagram above):

Component	Purpose in this application	Location in the DVSDK
GStreamer	Multimedia Framework	linux-devkit/arm-arago-linux-gnueabi/usr/lib

To construct your own pipelines there are examples of how to use the open source plugins in various places on the web including the GStreamer homepage. And to learn more about GStreamer-ti plugin architecture watch this online video and visit gstreamer.ti.com.

See the GStreamer Application Development Manual and the GStreamer 0.10 Core Reference Manual on how to write GStreamer applications.

Compiling an application

DVSDK Linux development kit includes the GStreamer development header files, libraries and package configs.

1. First, configure your cross compilation environment #Setting_up_cross_compilation_environment

2. Next, follow the typical GStreamer recommended method for compiling your application. e.g.

```
host $ cd <directory where your application is>
host $ gcc -o decode decode.c `pkg-config --libs --cflags gstreamer-0.10`
```

Additional Procedures

## Setting up cross compilation environment

To enable your application development, DVSDK comes with linux-devkit which contains package header, libraries and other package dependent information needed during development. Execute the following commands to configure your cross compilation environment

```
host $ source ${DVSDK}/linux-devkit/environment-setup
```

The above command will export cross compilation specific environment variables.

You will notice that the command will add **[linux-devkit]** to your bash prompt to indicate that you have exported the required cross compiler variables.

Rebuilding the DVSDK components

The DVSDK has provided a top level Makefile to allow the re-building of the various components within the DVSDK.

Note: The DVSDK component build environment is self contained and doesn't require the #Setting_up_cross_compilation_environment thus should be avoided to prevent possible build failures.

Rebuild the DVSDK components by first entering the DVSDK directory using:

```
host $ cd ${DVSDK}
```

The DVSDK makefile has a number of build targets which allows you to rebuild the DVSDK components. For a complete list execute:

```
host $ make help
```

Some of the components delivered in the DVSDK are not pre-built. The provided 'make clean' & 'make components' build targets are designed to clean and build all components (e.g. Linux Kernel, U-boot, CMEM, DMAI, etc.) for which a build is compulsory to begin application development. These components must first be cleaned and then rebuilt by the user before the user attempts to rebuild anything else. To do this, simply run

```
host $ make clean
host $ make components
```

After that, each of the build targets listed by 'make help' can then be executed using:

```
host $ make <target>_clean
host $ make <target>
host $ sudo make <target>_install
```

In order to install the resulting binaries on your target, execute one of the "install" targets using "sudo" privileges. Where the binaries are copied is controlled by the `EXEC_DIR` variable in `${DVSDK}/Rules.make`. This variable is set up to point to your NFS mounted target file system when you executed the DVSDK setup (`setup.sh`) script, but can be manually changed to fit your needs.

You can remove all components (including demos and examples) generated files at any time using:

```
host $ make clean
```

And you can rebuild all components and demos/examples using:

```
host $ make all
```

You can then install all the resulting target files using:

```
host $ sudo make install
```

Note: By default make install will override existing files and this can be controlled via modifying EXEC_DIR variable in Rules.make file.

Note: Booting newly build kernel requires you to run "depmod -a" command on target to regenerate new module dependencies for modprobe to work properly.

Creating your own Linux kernel image

The pre-built Linux kernel image (uImage) provided with the DVSDK is compiled with a default configuration. You may want to change this configuration for your application, or even alter the kernel source itself. This section shows you how to recompile the Linux kernel provided with the DVSDK, and shows you how to boot it instead of the default Linux kernel image.

1. If you haven't already done so, follow the instructions in #Setting_up_the_DVSDK to setup your build environment.

2. Recompile the kernel provided with the DVSDK by executing the following:

```
host $ cd ${DVSDK}
host $ make linux_clean
host $ make linux
host $ sudo make linux_install
```

3. You will need a way for the boot loader (u-boot) to be able to reach your new uImage. TFTP server has been setup in the #Setting_up_the_DVSDK section.

4. Copy your new uImage from the EXEC_DIR specified in the file ${DVSDK}/Rules.make to the tftpserver:

```
host $ cp ${HOME}/targetfs/boot/uImage /tftpboot/new_uImage
```

5. Run the u-boot script and follow the instructions. Select TFTP as your Linux kernel location and the file 'new_uImage' as your kernel image.

```
host $ ${DVSDK}/bin/setup-uboot-env.sh
```

6. Note that when you change your kernel, it is important to rebuild all the kernel modules supplied by the DVSDK sub-components. You can find a list of these modules under the directory /lib/modules/<kernel-version>/kernel/drivers/dsp/ (replace <*kernel-version*> with the version of the kernel applicable to your platform)

```
host $ ls ${HOME}/targetfs/lib/modules/<kernel-version>/kernel/drivers/dsp/
```

For each module that you see listed, you should go back to the host, rebuild it, and replace the file with the one from your EXEC_DIR. E.g. for cmemk.ko

```
host $ cd ${DVSDK}
host $ make cmem_clean
host $ make cmem
host $ sudo make cmem_install
```

You can also opt to re-build all the TI provided kernel modules (CMEM, etc.) and examples applications including the Linux kernel modules by issuing a **make all**. Then running the **make install** as described in the #Rebuilding_the_DVSDK_components.

8. After updating all modules, start minicom or Tera Term and power-cycle the board. The new kernel will now be loaded over TFTP from your Linux host.

9. Re-generate the kernel module dependency file on target by running the below command.

```
target $ depmod -a
```

Setting up Tera Term

Tera Term is a commonly used terminal program on Windows. If you prefer to use it instead of Minicom, you can follow these steps to set it up.

1. Download Tera Term from this location, and start the application.

2. In the menu select *Setup->General...* and set:

```
Default port: COM1
```

3. In the menu select *Setup->Serial Port...* and set the following:

```
Port:         COM1
Baud rate:    115200
Data:         8 bits
Parity:       none
Stop:         1 bit
Flow control: none
```

Integrating a new Codec in the DM3730 DVSDK

There are codecs available on the C64x+ Codec Page which are not included in the DVSDK because they require additional licenses. In order to use these codecs in the DVSDK environment they must be integrated in the DM3730 Codec Server. The following steps describe how to integrate a new codec into the DM3730 Codec Server package. The MP3 Decoder is used as an example but the information applies to any codec.

NOTE: That xx_xx_xx_xx should corresponded to the specific version associated with your DVSDK installation.

1. Download the MP3 decoder package from the Linux Download section located on the C64x+ Audio Codecs Page and install it.

```
host $ ./c64xplus_mp3dec_x_xx_xxx_production.bin
```

2. Extract the tar file from the directory the MP3 codec was installed.

```
host $ tar -xf dm6446_mp3dec_x_xx_xxx_production.tar
```

3. Make a copy of the DM3730 codec server folder `$(DVSDK)/codecs-omap3530_x_xx_xx_xx` and rename it:

```
host $ cd $(DVSDK)
host $ cp -r codecs-omap3530_xx_xx_xx_xx codecs-omap3530_x_xx_xx_xx_test
```

4. Update in top level Rules.make file to point to the directory of the newly copied codec folder:

```
host $ gedit Rules.make
```

5. Modify to following lines in the Rules.make as shown in *italics* below:

```
# Where the codecs are installed.
# CODEC_INSTALL_DIR=$(SDK_INSTALL_DIR)/codecs-omap3530_x_xx_xx_xx
CODEC_INSTALL_DIR=$(SDK_INSTALL_DIR)/codecs-omap3530_x_xx_xx_xx_test
```

6. Save and exit editor.

7. Copy the folder mp3dec directory from the from MP3 codec directory to the DVSDK codecs folder previously created and cd into the directory:

```
host $ cp -r \
dm6446_mp3dec_x_xx_xxx_production/packages/ti/sdo/codecs/mp3dec \
$(DVSDK)/codecs-omap3530_x_xx_xx_xx_test/packages/ti/sdo/codecs/.

host $ cd \
$(DVSDK)/codecs-omap3530_x_xx_xx_xx_test/packages/ti/sdo/codecs/mp3dec
```

8. Build all the codec packages by running "make codecs_clean" followed by "make codecs" in the $(DVSDK) folder. All the codec packages must be built in order for the Codec Engine GenServer Wizard to recognize them. If "make all" hasn't been previously executed as described in the #Rebuilding_the_DVSDK_components section, do so before running the next commands.

```
host $ cd $(DVSDK)
host $ make codecs_clean
host $ make codecs
```

9. The Codec Engine GenServer Wizard is a tool that is used to generate server packages. For information on GenServer Wizard go to The Codec Engine GenServer Wizard FAQ

To Launch the server wizard:

```
host $ cd $(DVSDK)/codecs-omap3530_x_xx_xx_xx_test/
host $ make -f Makefile.ce.genserver
```

10. From the wizard GUI open the file (File -> Open) `codecs-omap3530_x_xx_xx_xx_test/ti_sdo_server_cs_wizard.svrwiz`. This file includes the configuration of the original omap3530 codec server.

11. Set the Destination Directory: **$(DVSDK)/codecs-omap3530_x_xx_xx_xx_test/packages**

12. Set C6000 TI cgtools Directory to: **$(DVSDK)/cgt6x_x_x_xx**

13. The Search Path must be modified to pick up the correct codec packages and dependent components. Click on **Set Search Path** button and remove all the existing paths. Now add the following:

- $(DVSDK)/codecs-omap3530_x_xx_xx_xx_test/packages
- $(DVSDK)/codec-engine_x_xx_xx_xx/packages
- $(DVSDK)/xdais_x_xx_xx_xx/packages
- $(DVSDK)/framework-components_x_xx_xx_xx/packages

14. Refresh Codec list, MP3DEC should now be in the codec list. If MP3DEC is not in the list make sure that the Search Path to the codec packages has been updated properly in the previous step. Select all the codecs

15. Select Next and then Finish.

16. Select "Yes" to save the values entered in a new configuration file. Set the file name to:

```
ti_sdo_server_cs_wizard_test.svrwiz
```

Set the Save folder in to: $(DVSDK)/codecs_omap3530_x_xx_xx_xx_test

17. Replace files from
`$(DVSDK)/codecs-omap3530_x_xx_xx_xx_test/packages/ti/sdo/server/cs/package.bld,`

`codec.cfg, server.cfg, server.tcf` with the ones in the original Codec Server directory `$(DVSDK)/codecs-omap3530_x_xx_xx_xx/packages/ti/sdo/server/cs`

18. Add the following to codec.cfg:

```
    var MP3DEC = xdc.useModule('ti.sdo.codecs.mp3dec.ce.MP3DEC');

        MP3DEC.serverFxns = "MP3DEC_INBUFCACHEFLUSH";
        MP3DEC.stubFxns = "AUDDEC1_STUBS";
        MP3DEC.alg.watermark = false;
        MP3DEC.alg.codeSection = "DDR2";
        MP3DEC.alg.udataSection = "DDR2";
        MP3DEC.alg.dataSection = "DDR2";
```

Add the following to codec.cfg in the Server.algs array:

```
    {name: "mp3dec", mod: MP3DEC , threadAttrs: {
        stackMemId: 0, priority: Server.MINPRI + 3},
        groupId : 1,
    },
```

19. Remove config.bld from the `server/cs` directory

20. Build the codec server package by running "make codecs" from the top on the $(DVSDK) directory.

```
host $ cd $(DVSDK)
host $ make codecs
```

21. Test the new server with one of the DMAI sample apps.

How to create an SD card

This section explained the procedure required for creating SD card image for DM3730 and the steps has been verified on 2GB and 4GB SD cards.

1. Plug an SD card on Linux host machine.

2. Run dmesg command to check the device node. Triple check this to ensure you do not damage your HDD contents!

```
host $ dmesg
   [14365.272631] sd 6:0:0:1: [sdc] 3862528 512-byte logical blocks: (1.97 GB/1.84 GiB)
   [14365.310602] sd 6:0:0:1: [sdc] Assuming drive cache: write through
   [14365.325542] sd 6:0:0:1: [sdc] Assuming drive cache: write through
   [14365.325571]  sdc: sdc1 sdc2
```

In this example, SD card is detected on /dev/sdc.

3. Run mksdboot script installed in DVSDK as show below

```
host $ sudo ${DVSDK}/bin/mksdboot.sh --device /dev/sdc --sdk ${DVSDK}
```

Wait for script to complete. On successful completion, remove the SD card from the host PC.

NOTE: When creating a bootable SD card, the mksdboot script uses the pre-built root filesystem, kernel and bootloader images provided in the DVSDK installation.

4. Power OFF the DM3730 EVM.

5. Set the SW4 switch to boot from SD.

- SW4 = 00100111 (high to low, i.e. SW4.1 = 1)
- 1 = "On" position on the switch

6. Insert the SD card into the DM3730 EVM.

7. Power ON the EVM.

Note! If your flash already has a u-boot environment stored, this will get picked up even while booting from SD-card. If this is the case, halt the u-boot auto boot process and enter the following command to erase the entire NAND:

```
u-boot :> nand erase
```

Note! If you want to recreate the full SD card with a separate partition for the DVSDK installer execute the following:

```
host $ sudo ${DVSDK}/bin/mksdboot.sh --device /dev/sdc --sdk ${DVSDK} \
/path/to/dvsdk_dm3730-evm_4_xx_xx_xx_xx_setuplinux
```

This takes significant extra time so it's not part of the default instructions.

How to copy boot loaders to NAND flash

This section explains the procedure required for copying boot loaders (x-loader and u-boot) on NAND flash. The steps assumes that you have a working SD card image.

1. Boot the DM3730 EVM with SD card. See #How to create an SD card

2. Halt the u-boot auto boot process and enter the following commands to copy x-loader (i.e MLO) and u-boot from SD card to NAND flash.

```
u-boot :> nand erase
u-boot :> nandecc hw 2
u-boot :> mmc init
u-boot :> fatload mmc 0 0x81600000 MLO
u-boot :> nand write.i 0x81600000 0 20000
u-boot :> nandecc hw 2
u-boot :> fatload mmc 0 0x81600000 u-boot.bin
u-boot :> nand write.i 0x81600000 80000 40000
```

3. Power OFF the EVM and remove the SD card.

4. Set the SW4 switch to boot from NAND (1 = "On" position on the switch):

```
SW4 = 00101010 (high to low, i.e. SW4.1 = 0)
```

5. Power ON the EVM.

NOTE: Optionally, you can use windows based serial flasher tool (located in ${DVSDK}/host-tools/flash_utils/windows) for copying boot loaders into NAND flash. For more information about serial flasher usage, see http://processors.wiki.ti.com/index.php/Flash_v1.3_User_Guide.

GPLv3 Disclaimer

There are GPLv3 licensed software components contained within the this SDK on host side. The software manifest (software_manifest.htm) is located in the docs/ directory of the installed SDK. All GPLv3 components are contained in the SDK directory.

These GPLv3 components are provided for development purposes only and are intended to be removed before installing the application(s) code in your final product.

Additional Resources

PSP Documentation

http://processors.wiki.ti.com/index.php/AM35x-OMAP35x-PSP_04.02.00.07_Feature_Performance_Guide

http://processors.wiki.ti.com/index.php/AM35x-OMAP35x-PSP_04.02.00.07_Release_Notes

http://processors.wiki.ti.com/index.php/AM35x-OMAP35x-PSP_04.02.00.07_UserGuide

Graphics SDK Getting Started

http://processors.wiki.ti.com/index.php/OMAP35x_Graphics_SDK_Getting_Started_Guide

Wireless LAN, Bluetooth and Crypto

http://processors.wiki.ti.com/index.php/OMAP35x_Wireless_Connectivity_Downloads

http://processors.wiki.ti.com/index.php/Cryptography_Users_Guide

http://processors.wiki.ti.com/index.php/WLAN-BT_UserGuide

Matrix Application launcher

http://processors.wiki.ti.com/index.php/Matrix_Users_Guide

Flashing and Pin mux utilities

http://processors.wiki.ti.com/index.php/Flash_v1.3_User_Guide

http://processors.wiki.ti.com/index.php/Pin_Mux_Utility_for_ARM_MPU_Processors

Missleaneous

http://processors.wiki.ti.com/index.php/DVSDK_4.x_FAQ

http://processors.wiki.ti.com/index.php/Category:AM/DM37x

http://processors.wiki.ti.com/index.php/How_to_Find_the_Silicon_Revision_of_your_OMAP35x_or_AM/DM37x